PIT - Práctica 3: Detección de Actividad de Voz (VAD)

Alicia Lozano Díez y **Pablo Ramírez Hereza**

Estudiante: Gloria del Valle Cano

7 de marzo de 2022

```
import IPython
import itertools
import pickle

import librosa
import librosa.display
import numpy as np
import scipy.io.wavfile
import matplotlib.pyplot as plt

import torch
import torch.nn as nn
import torch.nn.functional as F

from tqdm.notebook import tqdm
from torch import optim
from google.colab import files
from IPython.core.pylabtools import figsize
from sklearn.metrics import confusion_matrix, accuracy_score
```

Objetivo

El objetivo de esta práctica es proporcionar una introducción al procesamiento de señales temporales de voz, y desarrollar de un detector de actividad de voz basado en redes neuronales recurrentes, en particular, LSTM.

Materiales

- Guión (.ipynb) de la práctica - Moodle
- Ejemplos de datos y etiquetas - Moodle
- Listas de entrenamiento y validación - Moodle
- Scripts de descarga de datos - Moodle
- Datos y etiquetas de entrenamiento * - One Drive (https://dauam-my.sharepoint.com/:u:/g/personal/alicia_lozano_uam_es/EdCueYU7BpNAuo6BawH8hJAB5rclap745BmsPzXgSPhsgw?e=Fh9adH)
- Datos y etiquetas de validación * - One Drive (https://dauam-my.sharepoint.com/:u:/g/personal/alicia_lozano_uam_es/EWBjWyX774pLhJc2ahr4zk0BtLvWt7YGhdMDDmGu-LcBNQ?e=sbgtjF)

CUIDADO: * Los datos proporcionados son de uso exclusivo para esta práctica. No tiene permiso para copiar, distribuir o utilizar el corpus para ningún otro propósito.

1. Introducción al procesamiento de señales temporales de voz

1.1. Descarga de ficheros de ejemplo

Primero vamos a descargar el audio de ejemplo de Moodle (**audio_sample.wav**) y ejecutar las siguientes líneas de código, que nos permitirán subir el archivo a Google Colab desde el disco local:

```
uploaded = files.upload()
```

Una vez cargado el fichero de audio, podemos escucharlo de la siguiente manera:

```
wav_file_name = "audio_sample.wav"
print(wav_file_name)
IPython.display.Audio(wav_file_name)
```

```
audio_sample.wav
```

```
<IPython.lib.display.Audio object>
```

1.2. Lectura y representación de audio en Python

A continuación vamos a definir ciertas funciones para poder hacer manejo de ficheros de audio en Python.

Comenzamos definiendo una función **read_recording** que leerá un fichero de audio WAV, normalizará la amplitud y devolverá el vector de muestras *signal* y su frecuencia de muestreo *fs*.

```
def read_recording(wav_file_name):
    fs, signal = scipy.io.wavfile.read(wav_file_name)  # read file
    signal = signal / max(abs(signal))  # normalizes amplitude
    duration_seconds = len(signal) / float(fs)  # duration (s)

    return fs, signal, duration_seconds
```

Si ejecutamos la función anterior para el fichero de ejemplo, podemos ver la forma en la que se carga dicho fichero de audio en Python. Así, podemos obtener la frecuencia de muestreo y la longitud del fichero en número de muestras:

```
fs, signal, duration_seconds = read_recording(wav_file_name)
print("Signal variable shape: " + str(signal.shape))
print("Sample rate: " + str(fs))
print("Duration (s): " + str(duration_seconds))
print("File length: " + str(len(signal)) + " samples")
```

```
Signal variable shape: (67072,)
Sample rate: 16000
Duration (s): 4.192
File length: 67072 samples
```

PREGUNTAS:

- ¿Como obtendría la duración de la señal en segundos?

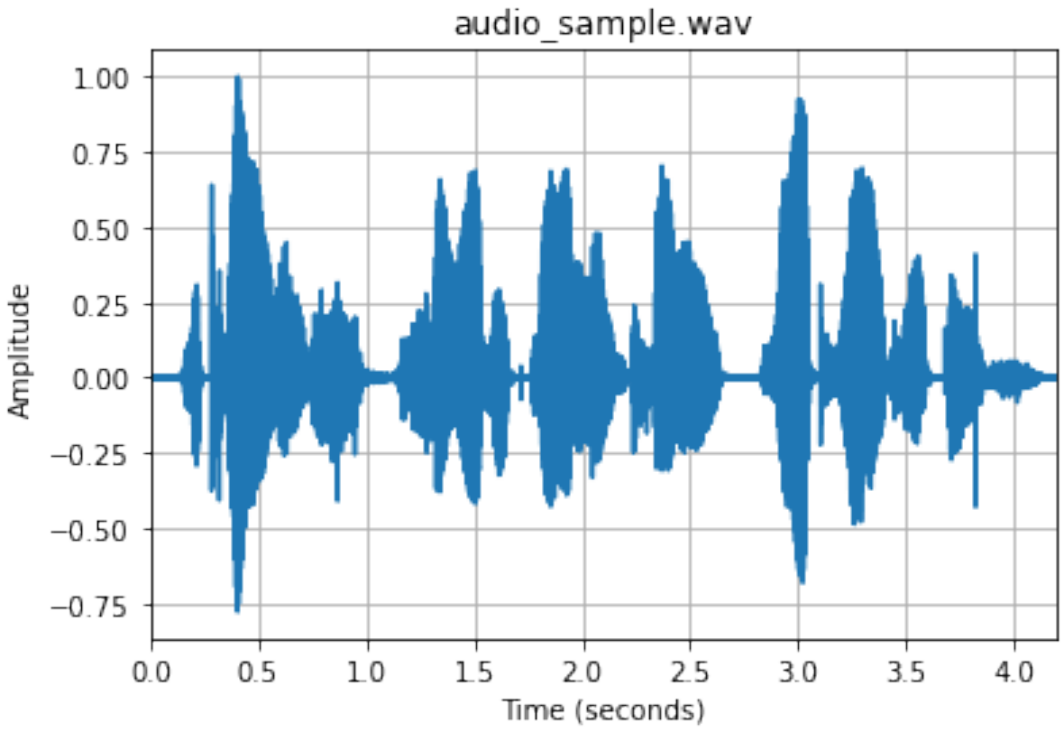
Para ello se ha añadido una variable extra de retorno en la función anterior, la cual se ha obtenido dividiendo el número de muestras de la señal por el sample rate del WAV. Vemos que tiene una duración de 4.192 segundos.

También podemos representar la señal y ver su forma de onda. Para ello, definimos la función **plot_signal** como sigue:

```
def plot_signal(signal, fs, ylabel="", title="", label=""):
    dur = len(signal) / fs
    step = 1.0 / fs
    t_axis = np.arange(0.0, dur, step)

    plt.plot(t_axis, signal, label=label)
    plt.xlim([0, dur])
    plt.ylabel(ylabel)
    plt.xlabel("Time (seconds)")
    plt.title(title)
    plt.grid(True)
```

Y utilizando la función anterior, obtenemos su representación (amplitud frente al tiempo):

```
plot_signal(signal, fs, "Amplitude", wav_file_name)
plt.show()
```



PREGUNTAS:

- Incluya en el informe la representación obtenida.

1.3. Representación de etiquetas de actividad de voz

En esta práctica, vamos a desarrollar un detector de actividad de voz, que determinará qué segmentos de la señal de voz son realmente voz y cuáles silencio.

Por ello, vamos a ver dos ejemplos de etiquetas *ground truth*, que corresponden al fichero de audio de ejemplo.

Primero, descargamos de Moodle las etiquetas de voz/silencio que están en los ficheros **audio_sample_labels_1.voz** y **audio_sample_labels_2.voz** y las cargamos en Google Colab como en el caso anterior.

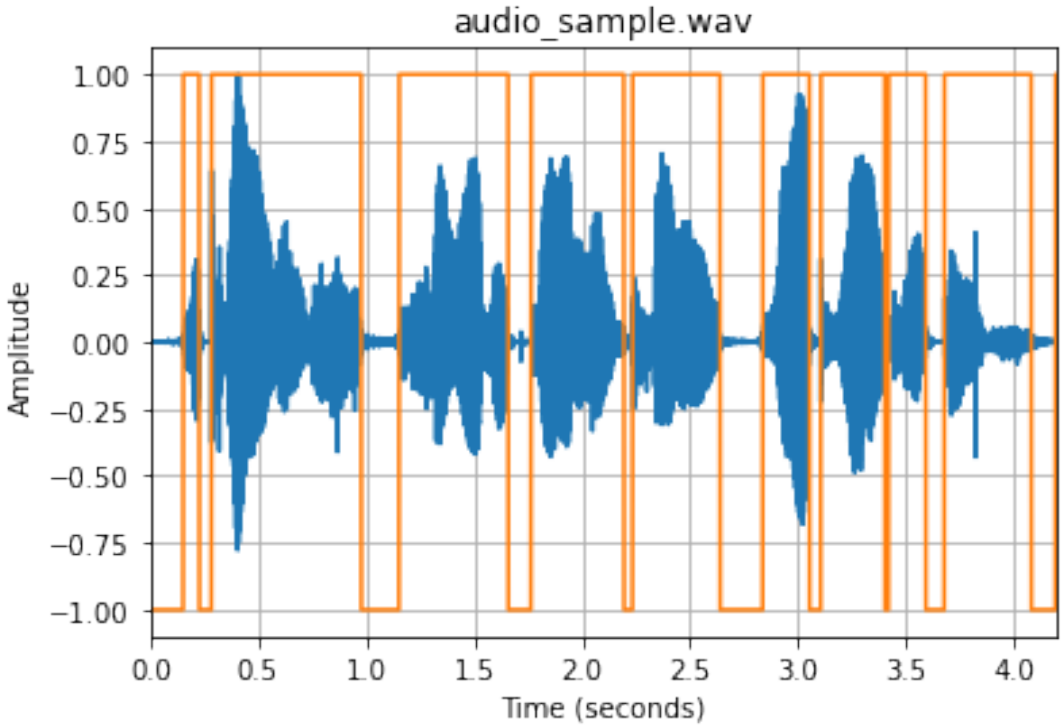
```
uploaded = files.upload()
```

Estas etiquetas están guardadas en ficheros de texto y podemos cargarlas en Python de la siguiente manera:

```
labels_file_name = "audio_sample_labels_1.voz"
voice_labels = np.loadtxt(labels_file_name)
```

Con el siguiente código, podemos representar la señal de voz así como sus etiquetas en la misma figura:

```
plot_signal(signal, fs)
plot_signal(voice_labels * 2 - 1, fs, "Amplitude", wav_file_name)
plt.show()
```



Las etiquetas de voz/silencio provienen de distintos detectores de actividad de voz.

```
print("Etiquetas iniciales: ", np.unique(voice_labels))
print("Etiquetas finales: ", np.unique(voice_labels * 2 - 1))
```

```
Etiquetas iniciales:  [0. 1.]
Etiquetas finales:  [-1.  1.]
```

PREGUNTAS:

- **¿Qué valores tienen las etiquetas? ¿Qué significan dichos valores?**

Vemos que la señal parece estar cuantizada con una binarización de 1 dígito, es decir, se toman los valores 0 y 1, que indican lo siguiente:

- 1: se contempla sonido.
- 0: se contemplan silencios.

Al representarse se quedan los valores -1 y 1.

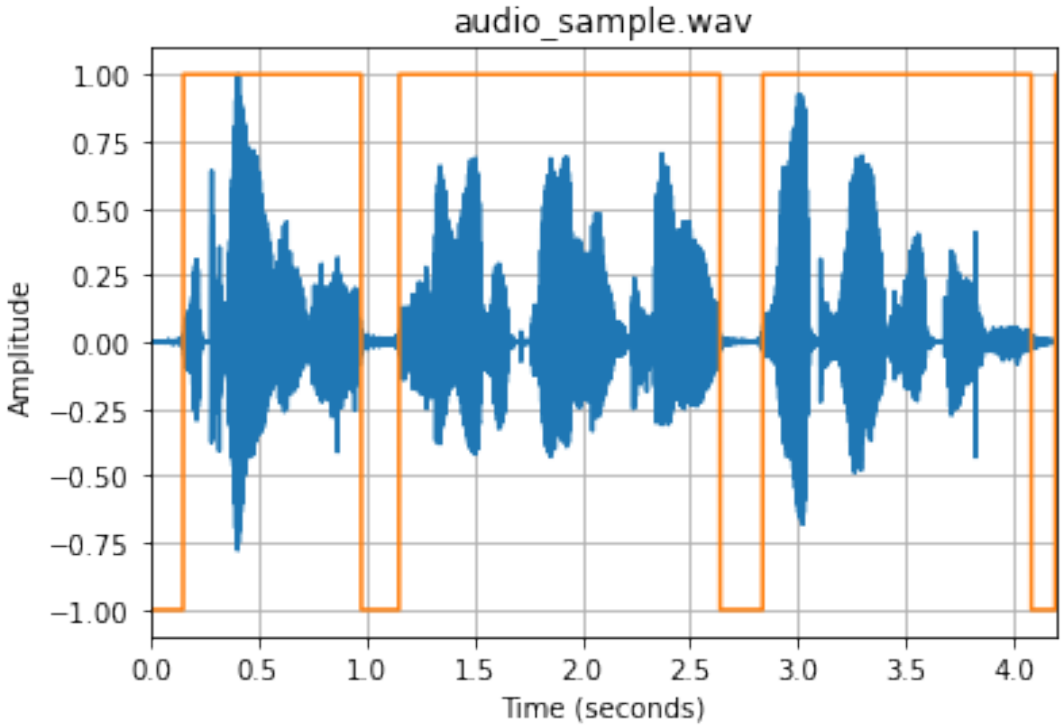
- **¿Por qué se representa _voice_labels*2-1_?**

Esto se realiza con el objetivo de normalizar la cuantización de la señal. Sin hacer nada de esto solo tendria en cuenta los valores mayores que cero, por lo que para normalizarla se

multiplica la escala por 2 (la amplitud sería 2) y se resta 1 para que quede por debajo, por tanto tenemos que las etiquetas son -1 cuando hay silencios y 1 cuando hay sonidos.

- **Represente la señal de voz junto con las etiquetas para ambos casos e incluya las figuras en el informe de la práctica. ¿Qué diferencias observas? ¿A qué se puede deber?**

```
labels_file_name = "audio_sample_labels_2.voz"
voice_labels = np.loadtxt(labels_file_name)
plot_signal(signal, fs)
plot_signal(voice_labels * 2 - 1, fs, "Amplitude", wav_file_name)
plt.show()
```



Podemos ver que la cantidad de silencios se considera aquí de manera diferente. Parece que los silencios están especificados de una manera diferente, considerando mayor umbral. Es posible, además, que la frecuencia de ambos detectores de actividad sea diferente o que uno sea más preciso que otro.

- **¿Qué cantidad de voz/silencio hay en cada etiquetado?**

Parece por tanto que hay más cantidad de voz en el segundo caso y menos silencios, mientras que en el primer caso detectar silencios es más sensible.

1.4. Extracción de características

En la mayoría de sistemas de reconocimiento de patrones, un primer paso es la extracción de características. Esto consiste, a grandes rasgos, en obtener una representación de los datos de entrada, que serán utilizados para un posterior modelado.

En nuestro caso, vamos pasar de la señal en crudo *"raw"* dada por las muestras (*signal*), a una secuencia de vectores de características que extraigan información a corto plazo de la misma y la representen. Esta sería la entrada a nuestro sistema de detección de voz basado en redes neuronales.

Para ver algunos ejemplos, vamos a utilizar la librería *librosa* (https://librosa.org/doc/latest/index.html).

Dentro de esta librería, tenemos funciones para extraer distintos tipos de características de la señal de voz, como por ejemplo el espectrograma en escala Mel (*melspectrogram*).

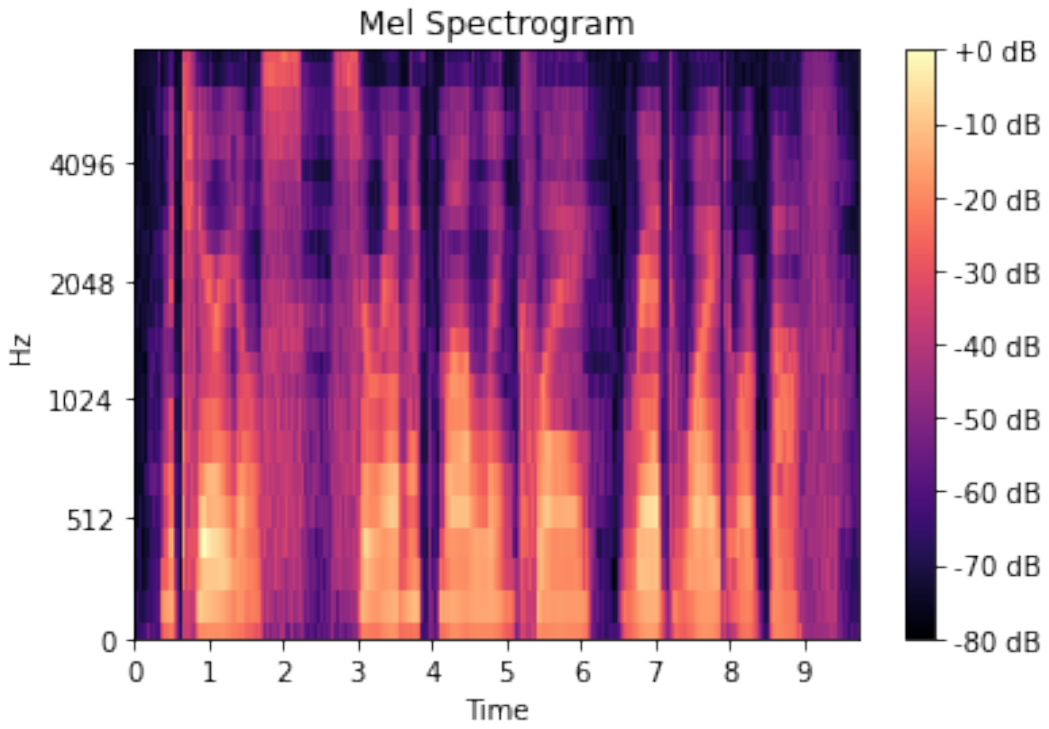
Estas características a corto plazo, se extraen en ventanas de unos pocos milisegundos con o sin solapamiento.

Un ejemplo sería el siguiente:

```
mel_spec = librosa.feature.melspectrogram(
    signal, fs, n_mels=23, win_length=320, hop_length=160
)

print(mel_spec.shape)
print(signal.shape)
```

```
(23, 420)
(67072,)
```

```
s_db = librosa.power_to_db(mel_spec, ref=np.max)
librosa.display.specshow(s_db, y_axis="mel", fmax=8000, x_axis="time")
plt.title("Mel Spectrogram")
plt.colorbar(format="%+2.0f dB");
```



PREGUNTAS:

- ¿Qué se obtiene de la función anterior?

Devuelve un espectograma en escala mel, que es una escala que se creó con la intención de detectar mejor las frecuencias altas, creando una unidad de tono tal que distancias iguales en el tono sonaran igualmente distantes para el oyente.

- ¿Qué significan los valores de los parámetros *win_length* y *hop_length*?
  - `win_length`: es el tamaño de ventana utilizado para el enventanado de la señal de audio.
  - `hop_length`: número de elementos espaciados entre ventanas sucesivas (sin solapar).
- ¿Qué dimensiones de *mel_spec* obtienes? ¿Qué significan? Vemos que se utiliza 23 features generadas (número de mels) por 420 ventanas totales.

De esta manera, podríamos obtener una parametrización de las señales para ser utilizadas como entrada a nuestra red neuronal.

Para los siguientes apartados, se proporcionan los vectores de características MFCC para una serie de audios que se utilizarán como conjunto de entrenamiento del modelo de VAD.

#2. Detector de actividad de voz (Voice Activity Detector, VAD)

2.1. Descarga de los datos de entrenamiento

Primero vamos a descargar la lista de identificadores de los datos de entrenamiento de la práctica.

Para ello, necesitaremos descargar de Moodle el fichero **training_VAD.lst**, y ejecutar las siguientes líneas de código, que nos permitirán cargar el archivo a Google Colab desde el disco local:

```
uploaded = files.upload()
```

A continuación cargamos los identificadores contenidos en el fichero en una lista en Python:

```
file_train_list = "training_VAD.lst"  # mat files containing data + labels
f = open(file_train_list, "r")
train_list = f.read().splitlines()
f.close()
```

Podemos ver algunos de ellos (los primeros 10 identificatores) de la siguiente forma:

```
print(train_list[:10])
```

```
['features_labs_1.mat', 'features_labs_10.mat',
'features_labs_100.mat', 'features_labs_101.mat',
'features_labs_102.mat', 'features_labs_103.mat',
'features_labs_104.mat', 'features_labs_105.mat',
'features_labs_106.mat', 'features_labs_107.mat']
```

Ahora, descargaremos de Moodle el fichero **data_download_onedrive_training_VAD.sh**, y ejecutaremos las siguientes líneas de código, que nos permitirán cargar el archivo a Google Colab desde el disco local:

```
uploaded = files.upload()
```

Para descargar el conjunto de datos desde One drive, ejecutamos el script cargado anteriormente de la siguiente manera:

```
!chmod 755 data_download_onedrive_training_VAD.sh
!./data_download_onedrive_training_VAD.sh
```

```
mkdir: cannot create directory ‘./data’: File exists
--2022-03-13 19:59:45--
https://dauam-my.sharepoint.com/:u:/g/personal/alicia_lozano_uam_es/
EdCueYU7BpNAuo6BawH8hJAB5rclap745BmsPzXgSPhsgw?download=1
Resolving dauam-my.sharepoint.com (dauam-my.sharepoint.com)...
13.107.136.9, 13.107.138.9
Connecting to dauam-my.sharepoint.com (dauam-my.sharepoint.com)|
13.107.136.9|:443... connected.
HTTP request sent, awaiting response... 302 Found
Location:
```

```
/personal/alicia_lozano_uam_es/Documents/PIT/training_VAD.zip
[following]
--2022-03-13 19:59:45--
https://dauam-my.sharepoint.com/personal/alicia_lozano_uam_es/Document
s/PIT/training_VAD.zip
Reusing existing connection to dauam-my.sharepoint.com:443.
HTTP request sent, awaiting response... 200 OK
Length: 3638232935 (3.4G) [application/x-zip-compressed]
Saving to: ‘EdCueYU7BpNAuo6BawH8hJAB5rclap745BmsPzXgSPhsgw?download=1’

EdCueYU7BpNAuo6BawH 100%[===================>]   3.39G  91.7MB/s    in
40s

2022-03-13 20:00:25 (87.6 MB/s) -
‘EdCueYU7BpNAuo6BawH8hJAB5rclap745BmsPzXgSPhsgw?download=1’ saved
[3638232935/3638232935]

replace training_VAD/features_labs_349.mat? [y]es, [n]o, [A]ll,
[N]one, [r]ename: y
replace training_VAD/features_labs_407.mat? [y]es, [n]o, [A]ll,
[N]one, [r]ename:
error:  invalid response [{ENTER}]
replace training_VAD/features_labs_407.mat? [y]es, [n]o, [A]ll,
[N]one, [r]ename: y
replace training_VAD/features_labs_361.mat? [y]es, [n]o, [A]ll,
[N]one, [r]ename: mv: cannot move './training_VAD' to
'./data/./training_VAD': Directory not empty
```

Este script descargará los datos de One Drive y los cargará en Google Colab, descomprimiéndolos en la carpeta **data/training_VAD**.

Podemos comprobar que los ficheros **.mat** se encuentran en el directorio esperado:

```
!ls data/training_VAD/ | head
```

```
features_labs_100.mat
features_labs_101.mat
features_labs_102.mat
features_labs_103.mat
features_labs_104.mat
features_labs_105.mat
features_labs_106.mat
features_labs_107.mat
features_labs_108.mat
features_labs_109.mat
```

2.2. Definición del modelo

Utilizando la librería Pytorch (https://pytorch.org/docs/stable/index.html), vamos a definir un modelo de ejemplo con una capa LSTM y una capa de salida. La capa de salida

estará formada por una única neurona. La salida indicará la probabilidad de voz/silencio utilizando una función *sigmoid*.

```
class Model_1(nn.Module):
    def __init__(self, feat_dim=20, hidden_size=256,
bidirectional=False):
        super(Model_1, self).__init__()
        self.lstm = nn.LSTM(
            feat_dim, hidden_size, batch_first=True,
bidirectional=bidirectional
        )
        linear_size = (
            hidden_size if not bidirectional else 2 * hidden_size
        )  # bidirectional case 256*2
        self.output = nn.Linear(linear_size, 1)

    def forward(self, x):

        out = self.lstm(x)[0]
        out = self.output(out)
        out = torch.sigmoid(out)

        return out.squeeze(-1)
```

PREGUNTAS:

- ¿Qué tamaño tiene la entrada a la capa LSTM?

Vemos que queda definido por feat_dim$=20$.

- ¿Cuántas unidades (celdas) tiene dicha capa LSTM?

El segundo parámetro de la LSTM es hidden_size, el cual vemos que es de 256.

- ¿Qué tipo de matriz espera la LSTM? Mirar la documentación y describir brevemente.

Se espera una matriz de la forma (N, L, H_{in}) donde N es referente al tamaño de batch, L es el tamaño de la secuencia y H_{in} es el tamaño de la entrada, ya que batch_first$=True$.

- Revisar la documentación de *torch.nn.LSTM* y describir brevemente los argumentos *batch_first*, *bidirectional* y *dropout*.

- batch_first: invierte las dos primeras dimensiones de la matriz de entrada, es decir, queda como (N, L, H_{in}) en vez de (L, N, H_{in}).

- dropout: en el caso de que sea mayor que cero se añade una capa de dropout tras cada etapa de la LSTM con la probabilidad pasada por parámetro.

- bidirectional: decide si la red es bidireccional o no (bi-LSTM).

- En este modelo, estamos utilizando una única neurona a la salida. ¿Hay alguna otra alternativa? ¿Se seguiría utilizando una función *sigmoid*?

Esta neurona se encarga de codificar la probabilidad de que haya voz en ese determinado instante. Podríamos utilizar otra neurona, sustituyendo la activación sigmoidal por una softmax, de forma que la primera neurona ofezca la probabilidad de silencio y la otra la de sonido.

- ¿Para qué sirve la función *forward* definida en la clase *Model_1*?

Computa la salida de la red a partir de la entrada, es decir, realiza el paso hacia adelante en la red neuronal. Para ello recibe como entrada un batch o segmento de características al que se le aplica la capa LSTM.

Una vez definida la clase, podemos crear nuestra instancia del modelo y cargarlo en la GPU con el siguiente código:

```
model = Model_1(feat_dim=20)
model = model.to(torch.device("cuda"))
print(model)
```

```
Model_1(
  (lstm): LSTM(20, 256, batch_first=True)
  (output): Linear(in_features=256, out_features=1, bias=True)
)
```

Nuestra variable *model* contiene el modelo, y ya estamos listos para entrenarlo y evaluarlo.

##2.3. Lectura y preparación de los datos para el entrenamiento

Como hemos visto anteriormente, nuestros datos están guardados en ficheros de Matlab (**.mat**). Cada uno de estos ficheros contiene una matriz **X** correspondiente a las secuencias de características MFCC (con sus derivadas de primer y segundo orden), y un vector **Y** con las etiquetas de voz/silencio correspondientes.

Veamos un ejemplo:

```
features_file = "data/training_VAD/features_labs_1.mat"
features = scipy.io.loadmat(features_file)["X"]
labels = scipy.io.loadmat(features_file)["Y"]

print(features.shape)
print(labels.shape)
```

```
(46654, 60)
(46654, 1)
```

PREGUNTAS: Elegir un fichero de entrenamiento y responder a las siguientes preguntas:

- ¿Qué tamaño tiene **features**? ¿Y **labels**?

Vemos que tenemos 46654 instantes con 60 características. Las etiquetas se corresponden con el valor de la señal en cada instante, siendo 46654.

- Una de las dimensiones de la **features** es 60, correspondiente a los 20 coeficientes MFCC concatenados con las derivadas de primer y segundo orden. ¿Con qué se corresponde la otra dimensión?

Es el número de ventanas tomadas al enventanar la señal.

El entrenamiento del modelo se va a realizar mediante descenso por gradiente (o alguna de sus variantes) basado en *batches*.

Para preparar cada uno de estos *batches* que servirán de entrada a nuestro modelo LSTM, debemos almacenar las características en secuencias de la misma longitud. El siguiente código lee las características (**get_fea**) y sus correspondientes etiquetas (**get_lab**) de un fragmento aleatorio del fichero de entrada.

```
def get_fea(segment, rand_idx):
    data = scipy.io.loadmat(segment)["X"]
    if data.shape[0] <= length_segments:
        start_frame = 0
    else:
        start_frame = np.random.permutation(data.shape[0] -
length_segments)[0]

    end_frame = np.min((start_frame + length_segments, data.shape[0]))
    rand_idx[segment] = start_frame
    feat = data[start_frame:end_frame, :20]  # discard D and DD, just
20 MFCCs
    return feat[np.newaxis, :, :]


def get_lab(segment, rand_idx):
    data = scipy.io.loadmat(segment)["Y"]
    start_frame = rand_idx[segment]
    end_frame = np.min((start_frame + length_segments, data.shape[0]))
    labs = data[start_frame:end_frame].flatten()
    return labs[np.newaxis, :]
```

PREGUNTAS: Analizar las funciones anteriores detenidamente y responder a las siguientes cuestiones:

- ¿De qué tamaño son los fragmentos que se están leyendo?

Por defecto se define a 300, con length_segments.

- ¿Para qué sirve *rand_idx*?

Sirve para tomar los diferentes batches, recorta y los toma de manera aleatoria.

2.4. Entrenamiento del modelo

Una vez definidas las funciones de lectura de datos y preparación del formato que necesitamos para la entrada a la red LSTM, podemos utilizar el siguiente código para entrenarlo.

Nota: utilizo este código para representar la matriz de confusión tras el entrenamiento.

```
def plot_confusion_matrix(
    cm, classes, normalize=False, title="Confusion matrix",
cmap=plt.cm.Blues
):
    """
    This function prints and plots the confusion matrix.
    Normalization can be applied by setting `normalize=True`.
    """
    if normalize:
        cm = cm.astype("float") / cm.sum(axis=1)[:, np.newaxis]
        print("Normalized confusion matrix")
    else:
        print("Confusion matrix, without normalization")

    plt.imshow(cm, interpolation="nearest", cmap=cmap)
    plt.title(title)
    plt.colorbar()
    tick_marks = np.arange(len(classes))
    plt.xticks(tick_marks, classes, rotation=45)
    plt.yticks(tick_marks, classes)

    fmt = ".2f" if normalize else "d"
    thresh = cm.max() / 2.0
    for i, j in itertools.product(range(cm.shape[0]),
range(cm.shape[1])):
        plt.text(
            j,
            i,
            format(cm[i, j], fmt),
            horizontalalignment="center",
            color="white" if cm[i, j] > thresh else "black",
        )

    plt.tight_layout()
    plt.ylabel("True label")
    plt.xlabel("Predicted label")
```

Creamos una función de train para más comodidad de uso en futuros entrenamientos de otros modelos.

```
def train_phase(segment_sets, model, path_in_feat):
    model.train()
```

```
    cache_loss = 0
    n_elems = 0
    eval_acc = 0
    cache_acc = 0
    true_labels, pred_labels = [], []

    for ii, segment_set in tqdm(enumerate(segment_sets)):
        rand_idx = {}
        optimizer.zero_grad()

        # Create training batches
        train_batch = np.vstack(
            [get_fea(path_in_feat + segment, rand_idx) for segment in
segment_set]
        )
        labs_batch = np.vstack(
            [
                get_lab(path_in_feat + segment,
rand_idx).astype(np.int16)
                for segment in segment_set
            ]
        )
        assert len(labs_batch) == len(
            train_batch
        )  # make sure that all frames have defined label

        # Shuffle the data and place them into Pytorch tensors
        shuffle = np.random.permutation(len(labs_batch))
        labs_batch = torch.tensor(
            labs_batch.take(shuffle, axis=0).astype("float32")
        ).to(torch.device("cuda"))
        train_batch = torch.tensor(
            train_batch.take(shuffle, axis=0).astype("float32")
        ).to(torch.device("cuda"))

        # Forward the data through the network
        outputs = model(train_batch)

        # Compute cost
        loss = criterion(outputs, labs_batch)

        # Backward step
        loss.backward()
        optimizer.step()
        cache_loss += loss.item()

        # Move logits and labels to CPU
        logits = outputs.detach().cpu().numpy().flatten()
        label_ids = labs_batch.cpu().numpy().flatten()
```

```
            # Calculate best predictions and true labels
            pred_labels += [1 if p >= 0.5 else 0 for p in logits]
            true_labels += list(label_ids)

    # Get accuracy score from sklearn
    accuracy = accuracy_score(true_labels, pred_labels)

    # Print also confusion matrix
    conf_matrix = confusion_matrix(true_labels, pred_labels)
    plt.figure()
    plot_confusion_matrix(
        conf_matrix, classes=["0", "1"], title="Confusion matrix,
without normalization"
    )
    plt.show()

    # Compute loss
    loss = cache_loss / len(train_batch)

    return loss, accuracy
```

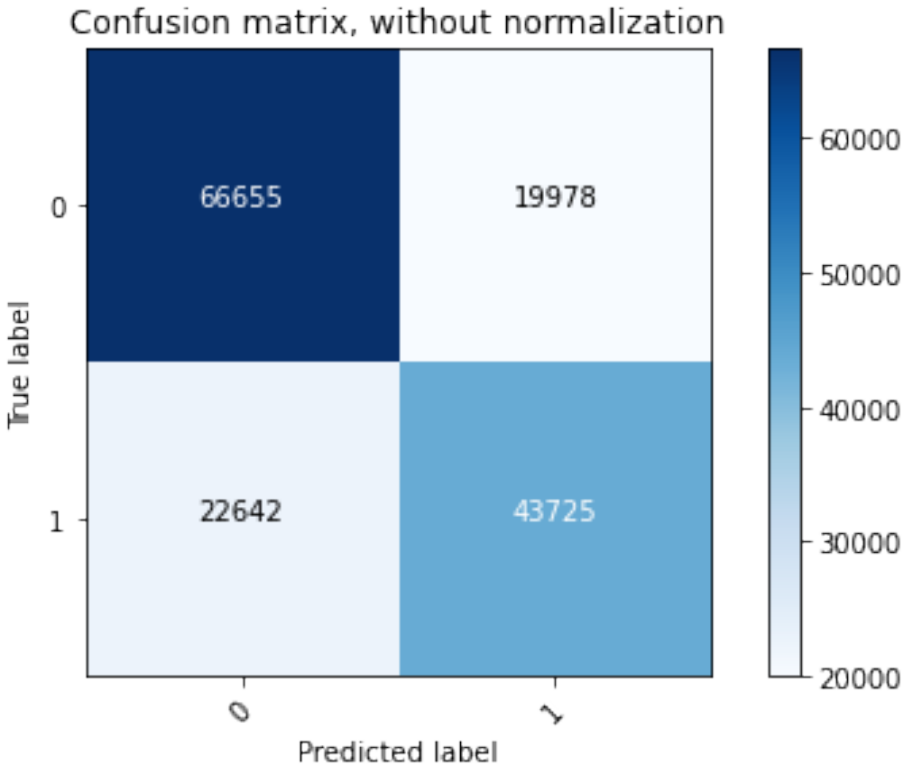
Entrenamos el modelo para cada época.

```
length_segments = 300
path_in_feat = "data/training_VAD/"

criterion = nn.BCELoss()
optimizer = optim.Adam(model.parameters(), lr=0.001)

batch_size = 51
segment_sets = np.array_split(train_list, len(train_list) /
batch_size)

max_iters = 5
accs, losses = [], []

for epoch in range(1, max_iters):
    print("Epoch: ", epoch)

    t_loss, t_acc = train_phase(segment_sets, model, path_in_feat)

    print("Train loss: {0}, Train acc: {1}".format(t_loss, t_acc))

    accs.append(t_acc)
    losses.append(t_loss)
```

```
Epoch:  1
```

{"version_major":2,"version_minor":0,"model_id":"ce7ab05cbdd9408ebc0cac881d56c5f0"}

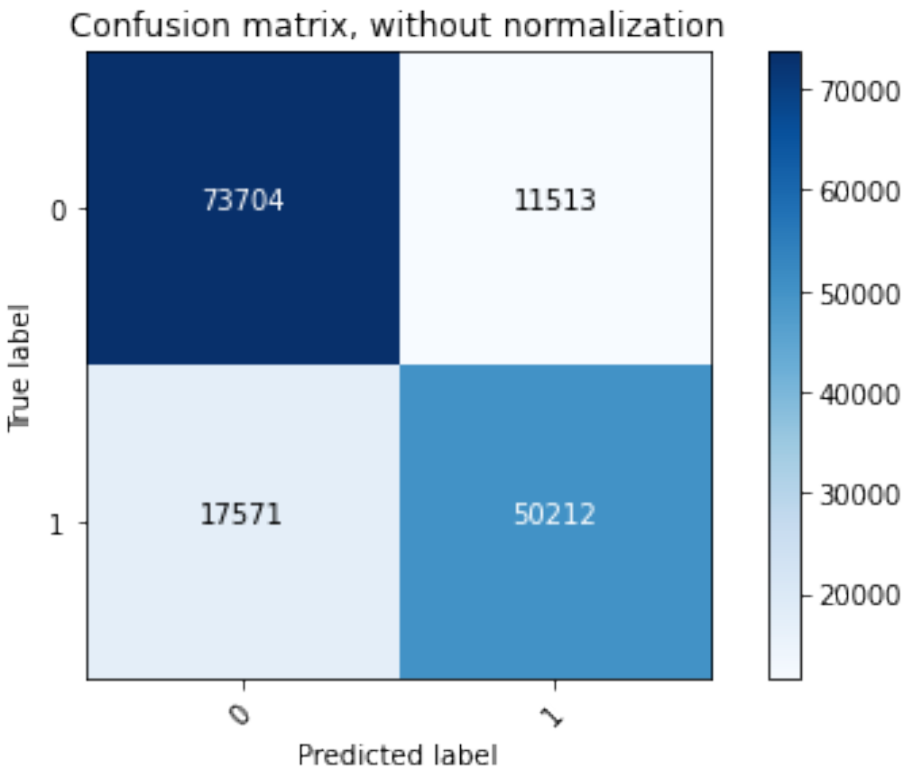
Confusion matrix, without normalization



Train loss: 0.11811170391007966, Train acc: 0.7214379084967321
Epoch: 2

{"version_major":2,"version_minor":0,"model_id":"b8e8399a48884f5097601d07ee170f92"}

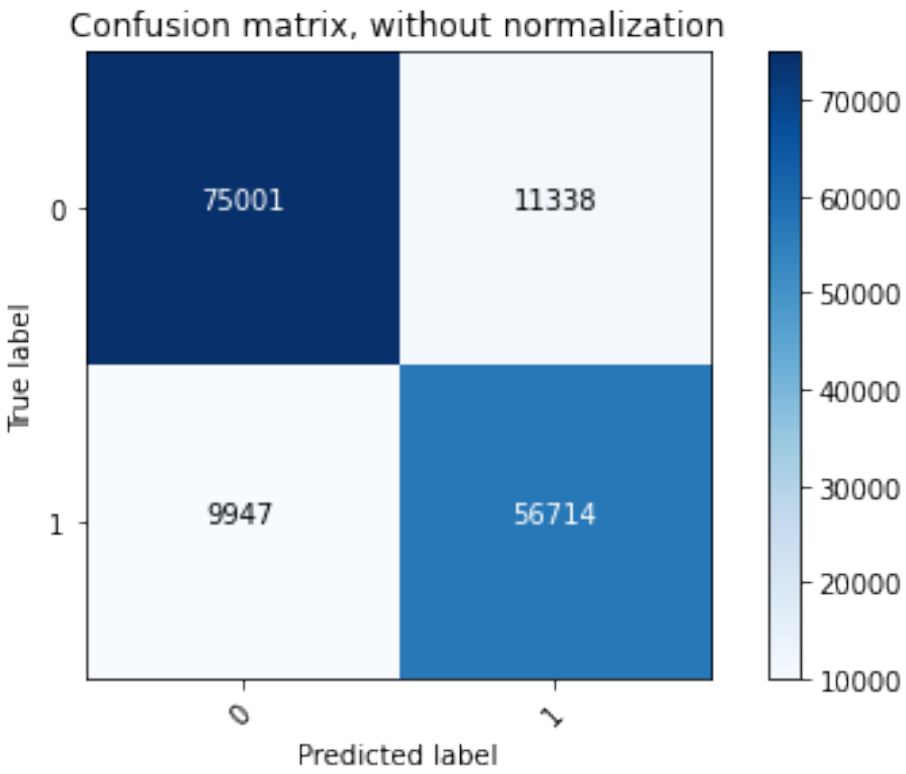
Confusion matrix, without normalization



Train loss: 0.08636569334011451, Train acc: 0.8099084967320261
Epoch: 3

{"version_major":2,"version_minor":0,"model_id":"34ba643d0052444ab3768
3fb194dbd2a"}

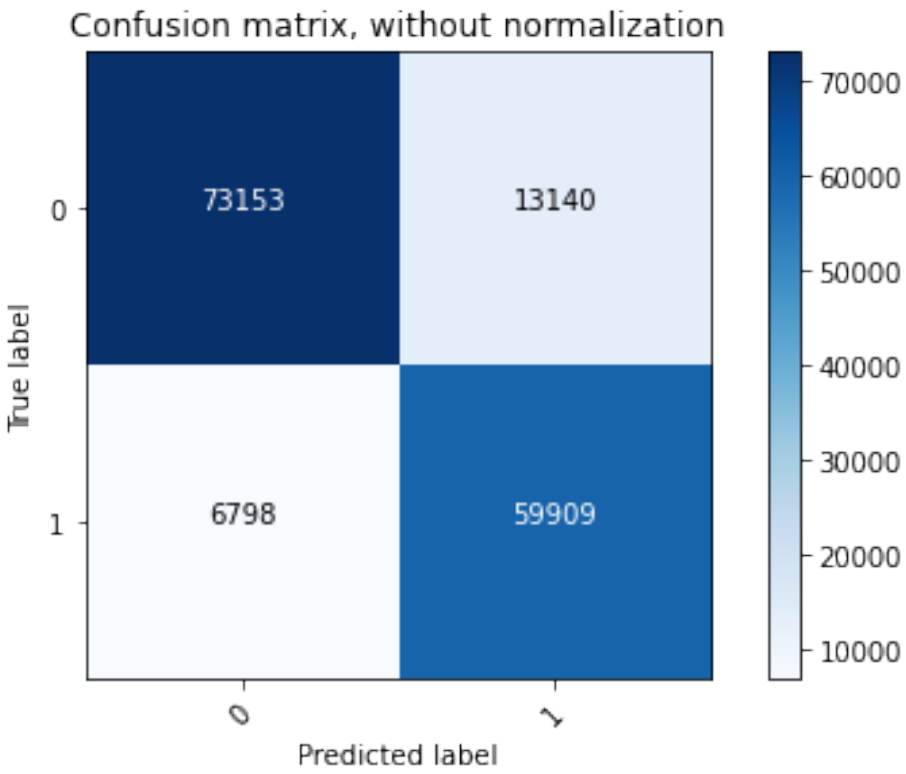
Confusion matrix, without normalization



Train loss: 0.06406454186813504, Train acc: 0.8608823529411764
Epoch: 4

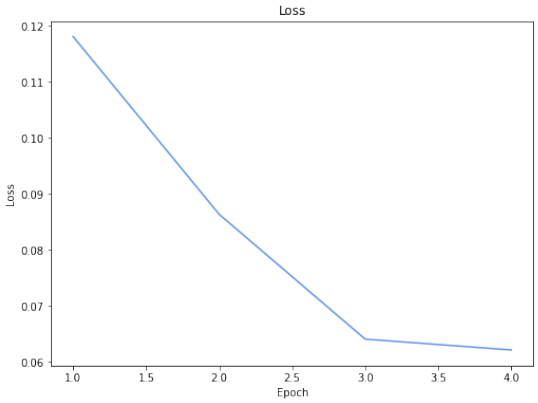
{"version_major":2,"version_minor":0,"model_id":"2d874f4af1af4b14ad76d3b81b3e6840"}

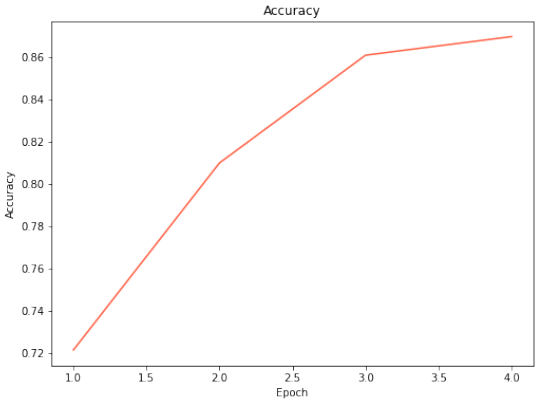
Confusion matrix, without normalization




```
Train loss: 0.062133130197431524, Train acc: 0.869686274509804
```

```python
def plot_results(losses, accs, epochs, title=""):

    fig, axs = plt.subplots(1, 2, figsize=(18, 6))

    axs[0].plot(epochs, losses, "cornflowerblue")
    axs[1].plot(epochs, accs, "tomato")

    axs[0].set_title("Loss")
    axs[1].set_title("Accuracy")

    axs[0].set_xlabel("Epoch")
    axs[0].set_ylabel("Loss")

    axs[1].set_xlabel("Epoch")
    axs[1].set_ylabel("Accuracy")

    plt.suptitle(title)

    plt.show()

plot_results(losses, accs, range(1, max_iters))
```






```
len(train_list) / 51
```

```
10.0
```

PREGUNTAS: Analizar el código anterior cuidadosamente y ejecutarlo. A continuación, responder a las siguientes cuestiones:

- ¿Qué función de coste se está optimizando? Describir brevemente con ayuda de la documentación.

Se ha utilizado la Entropía Cruzada Binaria (criterion).

- ¿Qué optimizador se ha definido?

Se ha utilizado Adam, con learning rate de 0.001.

- ¿Para qué se utiliza *batch_size*?

Para poder determinar el tamaño de los segmentos o batches.

- Describir brevemente la creación de los *batches*.

Se realizan con la función de numpy de `array_split`, tomando 510 valores (tamaño del conjunto de entrenamiento) entre 51, es decir, realiza particiones de 10 conjuntos de 51 características.

- ¿Qué línea de código realiza el *forward pass*?

(24) Computa los valores de la salida a partir de los de la entrada.

- ¿Qué línea de código realiza el *backward pass*?

(30) Computa los gradientes.

- ¿Cuántas iteraciones del algoritmo ha realizado? ¿Qué observa en la evolución de la función de coste?

Se han realizado 4 iteraciones.

- Añada al código el cálculo de la precisión o *accuracy*, de tal manera que se muestre por pantalla dicho valor en cada iteración (similar a lo que ocurre con el valor del coste *loss*). Copiar el código en el informe y describir brevemente.

Se ha añadido en el código y se han realizado unos plots para ver mejor los resultados en cada época. Se observa cómo el valor de la pérdida es cada vez menor, mientras que el valor de la accuracy crece hasta el valor 0.875.

- ¿Qué valor de coste y *accuracy* obtiene? ¿Cómo se puede mejorar?

Se obtiene finalmente un valor de coste de 0.0621 y una accuracy de 0.869. Tal vez se podrían realizar más épocas o cambiar el tamaño del batch para vez si los resultados mejoran. Además se podría investigar si el uso de una bi-LSTM podría mejorar los resultados.

2.5. Evaluación del modelo: un único fichero de test

Una vez entrenado el modelo, vamos a evaluarlo en un ejemplo en concreto.

Descargue de Moodle el fichero **audio_sample_test.wav**, con sus correspondientes características y etiquetas **audio_sample_test.mat** y evalúe el rendimiento en el mismo.

```
features_file = "audio_sample_test.mat"
features = scipy.io.loadmat(features_file)["X"]
labels = scipy.io.loadmat(features_file)["Y"]

print(features.shape)
print(labels.shape)
```

```
(57777, 60)
(57777, 1)
```

```
# Get features
features = features[:, :20]

# Move features to tensor
tensor_data =
torch.tensor([features]).to(torch.device("cuda")).float()

# Forward the data through the network
model.eval()
with torch.no_grad():
    outputs = model.forward(tensor_data)

# Move logits and labels to cpu
outputs = outputs.cpu().detach().numpy().flatten()
labels = labels.flatten()

# Compute accuracy
preds_labels = np.round(outputs)
```

```
test_acc = np.mean(preds_labels == labels)
print("Test accuracy: {}".format(test_acc))
```

```
Test accuracy: 0.9014313654222268
```

PREGUNTAS:

- Incluya en el informe de la práctica el código que ha utilizado para evaluar dicho fichero.

Hecho.

- ¿Cuál es el *accuracy* obtenido para el fichero **audio_sample_test**?

Se ha obtenido un valor de 0.901, lo que parece bastante alto.

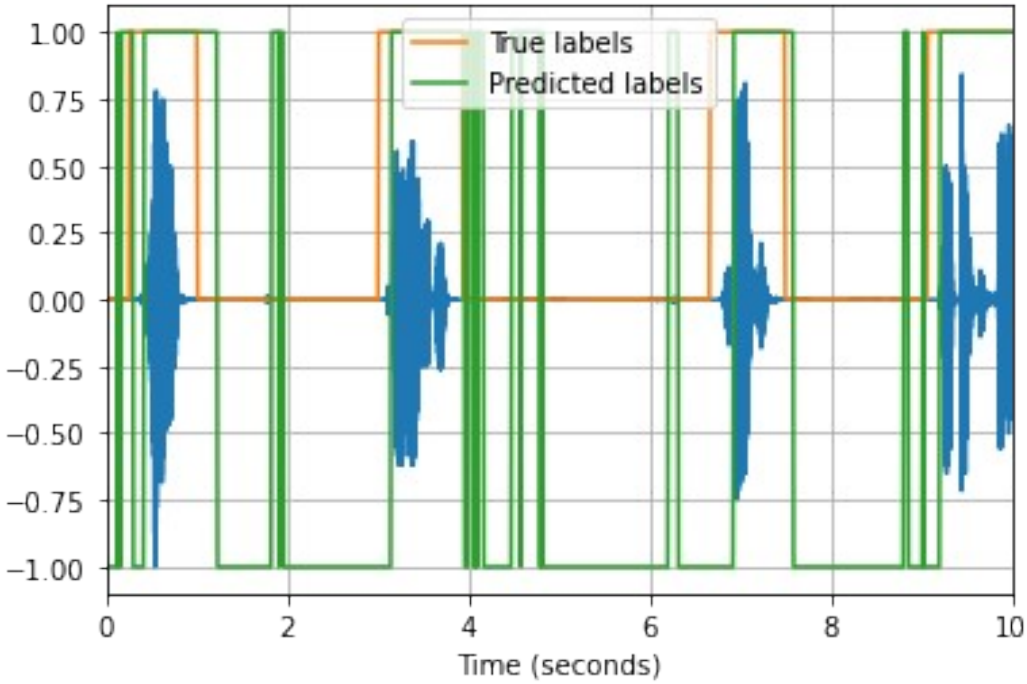
- Represente 10 segundos de dicho audio, así como sus etiquetas de *ground_truth* y las obtenidas con su modelo. Incluya dicha gráfica en el informe y comente brevemente el resultado. Visualmente, ¿es bueno el modelo?
- Escuche el audio y comente cualitativamente cómo es de bueno o malo el modelo.

Estas dos preguntas se contestan a continuación.

```
wav_file_name = "audio_sample_test.wav"
fs, signal, duration_seconds = read_recording(wav_file_name)
print("Signal variable shape: " + str(signal.shape))
print("Sample rate: " + str(fs))
print("Duration (s): " + str(duration_seconds))
print("File length: " + str(len(signal)) + " samples")

segment = 1
n_seconds = 10
n_labels_segment = n_seconds * 100
fs_labels = 100.0

signal_segment = signal[segment * n_seconds * fs : (segment + 1) *
n_seconds * fs]
labels_segment = labels[segment * n_labels_segment : (segment + 1) *
n_labels_segment]
pred_labels_segment = preds_labels[
    segment * n_labels_segment : (segment + 1) * n_labels_segment
]

print(f"N samples from second {n_seconds}:", len(signal_segment))
```

```
Signal variable shape: (4800160,)
Sample rate: 8000
Duration (s): 600.02
File length: 4800160 samples
N samples from second 10: 80000
```

```
plot_signal(signal_segment, fs)
plot_signal(labels_segment, fs_labels, label="True labels")
plot_signal(pred_labels_segment * 2 - 1, fs_labels, label="Predicted
labels")
plt.legend()
plt.show()
```



Parece que las etiquetas de predicción están ligeramente desplazadas pero cumplen con las reales.

```
print(wav_file_name)
IPython.display.Audio(wav_file_name)
```

```
audio_sample_test.wav
```

```
<IPython.lib.display.Audio object>
```

Parece una conversación telefónica en la cual existen partes de habla y partes de silencio. Aunque parece que en momentos detecta como sonido partes de ruido, en general lo hace bien.

2.6. Evaluación del modelo: conjunto de validación

Ahora vamos a evaluar el rendimiento del modelo anterior sobre un conjunto de validación (del que conocemos sus etiquetas).

Para este conjunto de datos, descargaremos la lista de identificadores **valid_VAD.lst** de Moodle, así como el fichero de descarga de datos **data_download_onedrive_valid_VAD.sh**:

```
!chmod 755 data_download_onedrive_valid_VAD.sh
!./data_download_onedrive_valid_VAD.sh
```

```
mkdir: cannot create directory ‘./data’: File exists
--2022-03-13 20:09:31--
https://dauam-my.sharepoint.com/:u:/g/personal/alicia_lozano_uam_es/
EWBjWyX774pLhJc2ahr4zk0BtLvWt7YGhdMDDmGu-LcBNQ?download=1
Resolving dauam-my.sharepoint.com (dauam-my.sharepoint.com)...
13.107.136.9, 13.107.138.9
Connecting to dauam-my.sharepoint.com (dauam-my.sharepoint.com)|
13.107.136.9|:443... connected.
HTTP request sent, awaiting response... 302 Found
Location: /personal/alicia_lozano_uam_es/Documents/PIT/valid_VAD.zip
[following]
--2022-03-13 20:09:32--
https://dauam-my.sharepoint.com/personal/alicia_lozano_uam_es/Document
s/PIT/valid_VAD.zip
Reusing existing connection to dauam-my.sharepoint.com:443.
HTTP request sent, awaiting response... 200 OK
Length: 508492067 (485M) [application/x-zip-compressed]
Saving to: ‘EWBjWyX774pLhJc2ahr4zk0BtLvWt7YGhdMDDmGu-LcBNQ?download=1’

EWBjWyX774pLhJc2ahr 100%[===================>] 484.94M  92.7MB/s    in
5.2s

2022-03-13 20:09:37 (93.8 MB/s) -
‘EWBjWyX774pLhJc2ahr4zk0BtLvWt7YGhdMDDmGu-LcBNQ?download=1’ saved
[508492067/508492067]

mv: cannot move './valid_VAD' to './data/./valid_VAD': Directory not
empty
```

```
file_val_list = "valid_VAD.lst"
f = open(file_val_list, "r")
val_list = f.read().splitlines()
f.close()
```

Escriba ahora el código necesario para evaluar el modelo anterior en el conjunto de datos de validación, para su última época.

Tenga en cuenta que si quiere realizar el forward para todos los datos de validación de una vez, necesitará que todas las secuencias sean de la misma longitud. Como aproximación, puede escoger unos pocos segundos de cada fichero como se hace en el entrenamiento.

```
def eval_phase(segment_sets, model, path_in_feat):
    model.eval()
    n_elems = 0
    cache_loss = 0
    eval_acc = 0
    cache_acc = 0
    n = 0
```

```
    with torch.no_grad():
        for ii, segment_set in tqdm(enumerate(segment_sets)):
            rand_idx = {}
            optimizer.zero_grad()

            # Create validation batches
            val_batch = np.vstack(
                [get_fea(path_in_feat + segment, rand_idx) for segment
in segment_set]
            )
            labs_batch = np.vstack(
                [
                    get_lab(path_in_feat + segment,
rand_idx).astype(np.int16)
                    for segment in segment_set
                ]
            )
            assert len(labs_batch) == len(
                val_batch
            )  # make sure that all frames have defined label

            # Place them into Pytorch tensors
            labs_batch =
torch.tensor(labs_batch.astype("float32")).to(
                torch.device("cuda")
            )
            val_batch = torch.tensor(val_batch.astype("float32")).to(
                torch.device("cuda")
            )

            # Forward the data through the network
            preds = model.forward(val_batch)

            # Compute cost
            loss = criterion(preds, labs_batch)

            # Loss step and acc
            cache_loss += loss.item()
            cache_acc += (torch.eq(preds >= 0.5, labs_batch)).sum()
            n += np.prod(preds.shape[:2])

    v_loss = cache_loss / len(segment_sets)
    v_acc = cache_acc / n

    return v_loss, v_acc
```

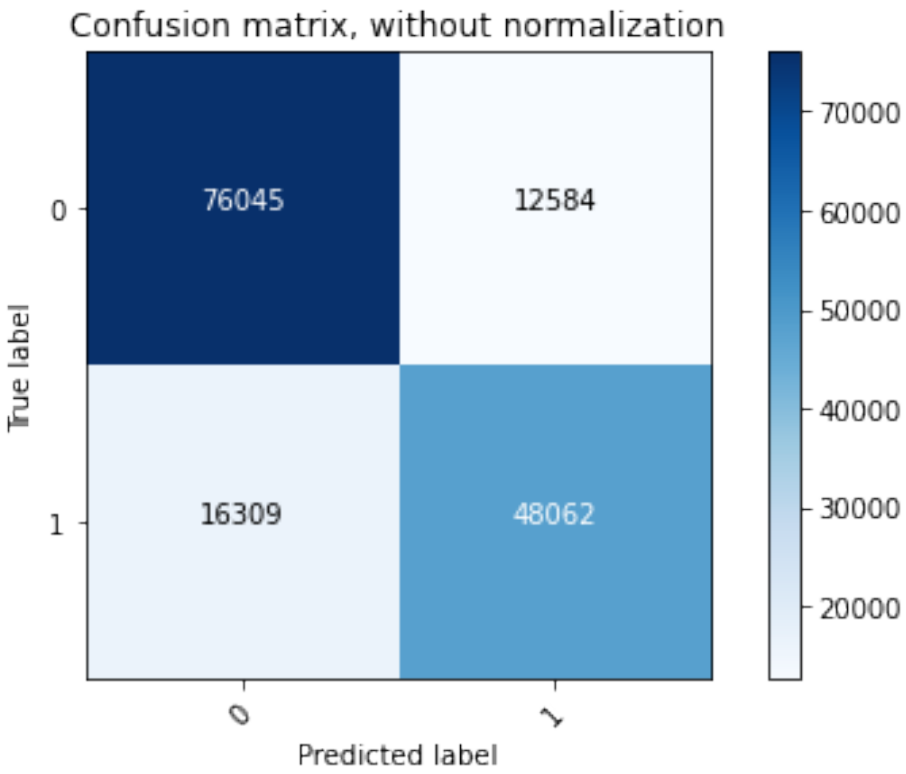
Entrenamos y validamos el modelo.

```
length_segments = 300

path_in_feat_train = "data/training_VAD/"
path_in_feat_val = "data/valid_VAD/"

criterion = nn.BCELoss()
optimizer = optim.Adam(model.parameters(), lr=0.001)

batch_size = 51
segment_sets_train = np.array_split(train_list, len(train_list) /
batch_size)
segment_sets_val = np.array_split(val_list, len(train_list) /
batch_size)

max_iters = 5

accs1, losses1 = [], []
accs2, losses2 = [], []

for epoch in range(1, max_iters):
    print("Epoch: ", epoch)

    t_loss, t_acc = train_phase(segment_sets_train, model,
path_in_feat_train)
    v_loss, v_acc = eval_phase(segment_sets_val, model,
path_in_feat_val)

    print("Train loss: {0}, Train acc: {1}".format(t_loss, t_acc))
    print("Val loss: {0}, Val acc: {1}".format(v_loss, v_acc))

    accs1.append(t_acc)
    accs2.append(v_acc.cpu())
    losses1.append(t_loss)
    losses2.append(v_loss)
```

```
Epoch:  1
```


```
{"version_major":2,"version_minor":0,"model_id":"e13b25f2799b4c05a2b8d
a1389cc51e8"}
```


```
Confusion matrix, without normalization
```

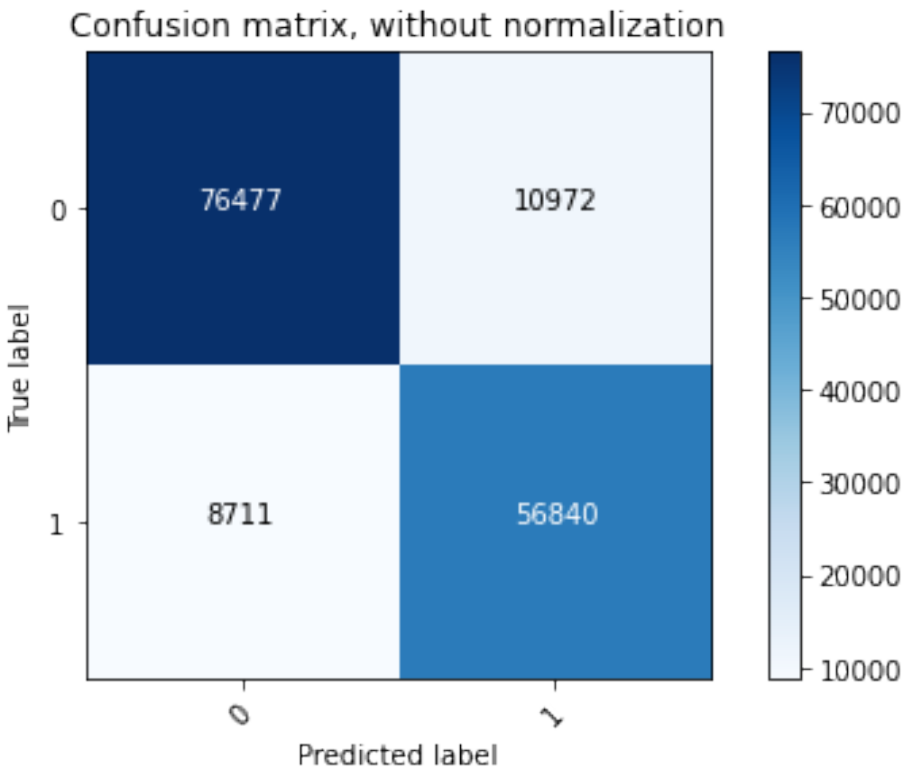


{"version_major":2,"version_minor":0,"model_id":"3829b14351ee491991f1dd9147c4aa71"}

Train loss: 0.0866928731693941, Train acc: 0.811156862745098
Val loss: 0.35542639791965486, Val acc: 0.8420370221138
Epoch: 2

{"version_major":2,"version_minor":0,"model_id":"87ef9fcb05ce478f9ef09de5b2fb38f3"}

Confusion matrix, without normalization

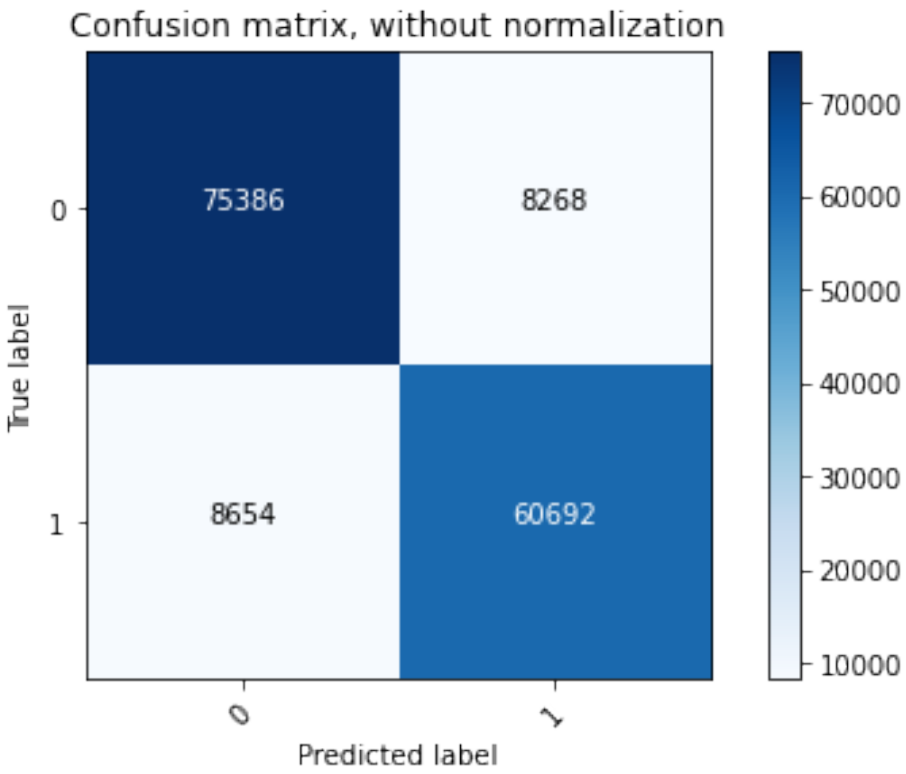


{"version_major":2,"version_minor":0,"model_id":"5c6aa8437544402ab3e7e6fe2b4cb802"}

Train loss: 0.061957091093063354, Train acc: 0.8713529411764706
Val loss: 0.31674617528915405, Val acc: 0.8835185170173645
Epoch: 3

{"version_major":2,"version_minor":0,"model_id":"4047d42ccad542e09e22fef582f6e626"}

Confusion matrix, without normalization

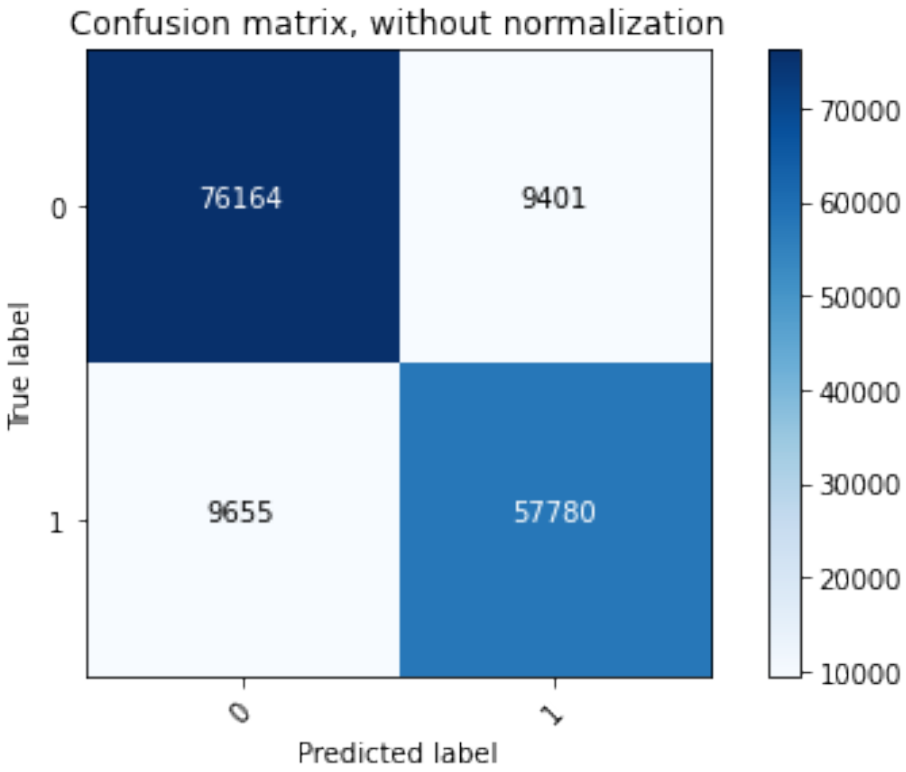


{"version_major":2,"version_minor":0,"model_id":"bbbd0ccef536422d9dfe58ebc42d74b5"}

Train loss: 0.05915868077792373, Train acc: 0.8893986928104575
Val loss: 0.33240152299404147, Val acc: 0.8680555820465088
Epoch: 4

{"version_major":2,"version_minor":0,"model_id":"f825330073cd4cd182ec60ede908ae34"}

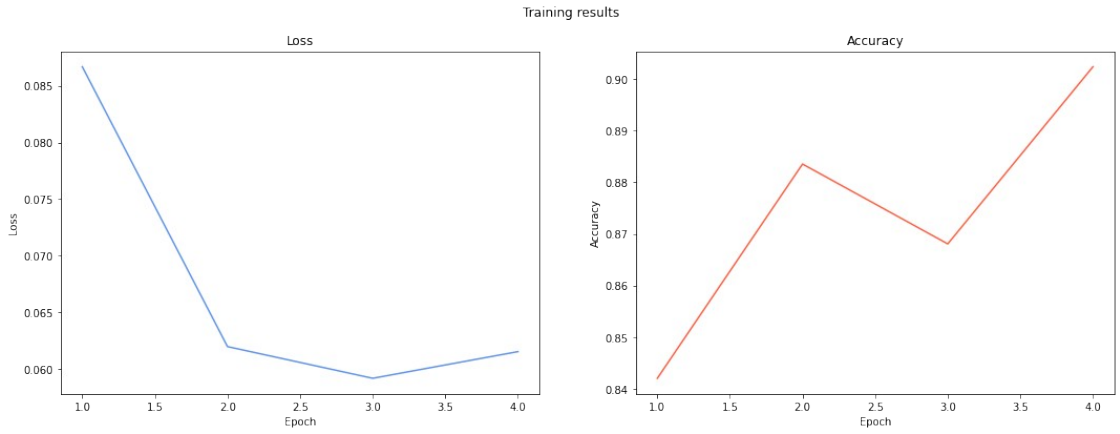
Confusion matrix, without normalization



{"version_major":2,"version_minor":0,"model_id":"6cf30c3f41d747c5b3f5e9ad172f48c2"}

```
Train loss: 0.06151433581230687, Train acc: 0.8754509803921569
Val loss: 0.259148907661438, Val acc: 0.9023610949516296
```

```
plot_results(losses1, accs2, range(1, max_iters), title="Training results")
```



PREGUNTAS:

- Incluya en la memoria de la práctica el código utilizado, incluyendo los valores de cualquier parámetro de configuración utilizado (por ejemplo, el número de épocas

de entrenamiento realizadas). Hecho, se ha mantenido el minsmo número de épocas para comparar los resultados.

- ¿Qué rendimiento (loss y accuracy) obtiene con este modelo (*Model_1*) en entrenamiento y en validación? Se ha obtenido una puntuación de 0.875 en entrenamiento y un 0.902 en validación.

3. Comparación de modelos

3.1. Redes LSTM bidireccionales

En este apartado, vamos a partir del modelo inicial (*Model_1*) y modificarlo para que la capa LSTM sea bidireccional (*Model_1B*).

Entrene el nuevo modelo y compare el resultado con el modelo inicial.

```
model = Model_1(feat_dim=20, bidirectional=True)
model = model.to(torch.device("cuda"))
print(model)
```

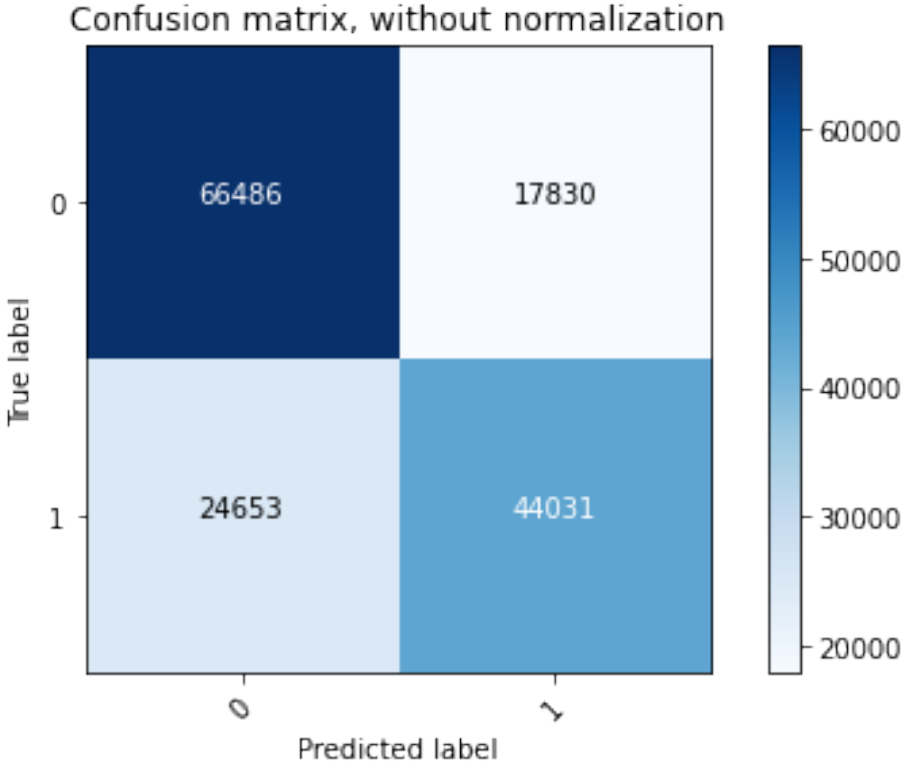
```
Model_1(
  (lstm): LSTM(20, 256, batch_first=True, bidirectional=True)
  (output): Linear(in_features=512, out_features=1, bias=True)
)
```

```
length_segments = 300

path_in_feat_train = "data/training_VAD/"
path_in_feat_val = "data/valid_VAD/"

criterion = nn.BCELoss()
optimizer = optim.Adam(model.parameters(), lr=0.001)

batch_size = 51
segment_sets_train = np.array_split(train_list, len(train_list) /
batch_size)
segment_sets_val = np.array_split(val_list, len(train_list) /
batch_size)

max_iters = 5

accs1, losses1 = [], []
accs2, losses2 = [], []

for epoch in range(1, max_iters):
    print("Epoch: ", epoch)

    t_loss, t_acc = train_phase(segment_sets_train, model,
path_in_feat_train)
```

```
    v_loss, v_acc = eval_phase(segment_sets_val, model,
path_in_feat_val)

    print("Train loss: {0}, Train acc: {1}".format(t_loss, t_acc))
    print("Val loss: {0}, Val acc: {1}".format(v_loss, v_acc))

    accs1.append(t_acc)
    accs2.append(v_acc.cpu())
    losses1.append(t_loss)
    losses2.append(v_loss)

Epoch:  1
```

{"version_major":2,"version_minor":0,"model_id":"fe70521b09fc4c58af32145c4fd9ca77"}

```
Confusion matrix, without normalization
```

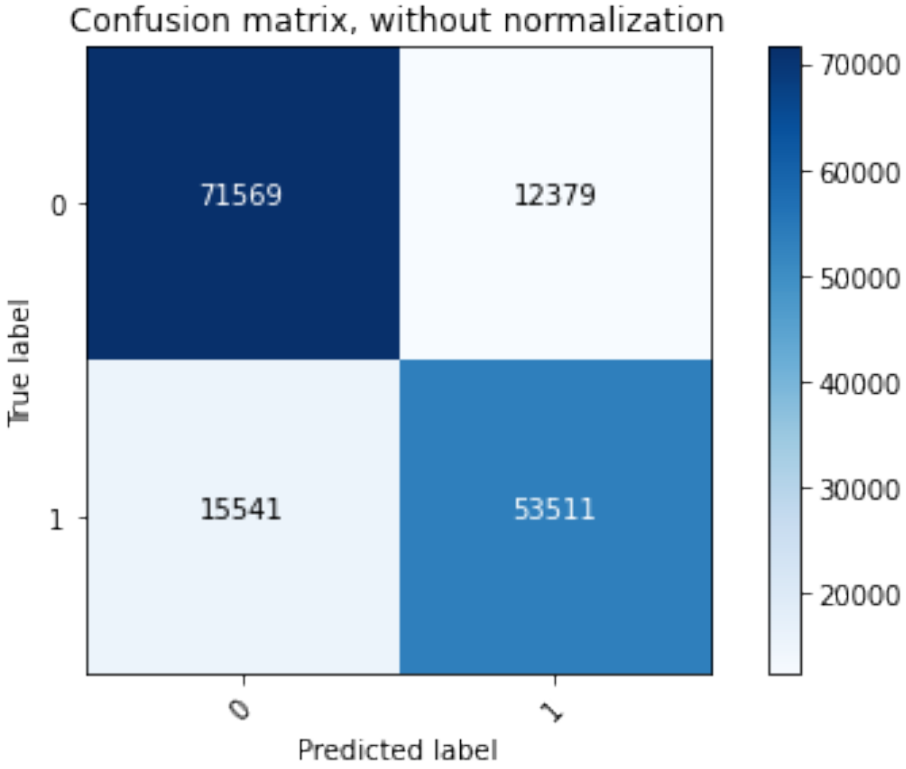


{"version_major":2,"version_minor":0,"model_id":"072fc3abee6a4ec0a122361d78707702"}

```
Train loss: 0.11591428752038993, Train acc: 0.7223333333333334
Val loss: 0.4772031337022781, Val acc: 0.8015277981758118
Epoch:  2
```

{"version_major":2,"version_minor":0,"model_id":"34eb6b4eb37a40a382b19998c068e5da"}

Confusion matrix, without normalization

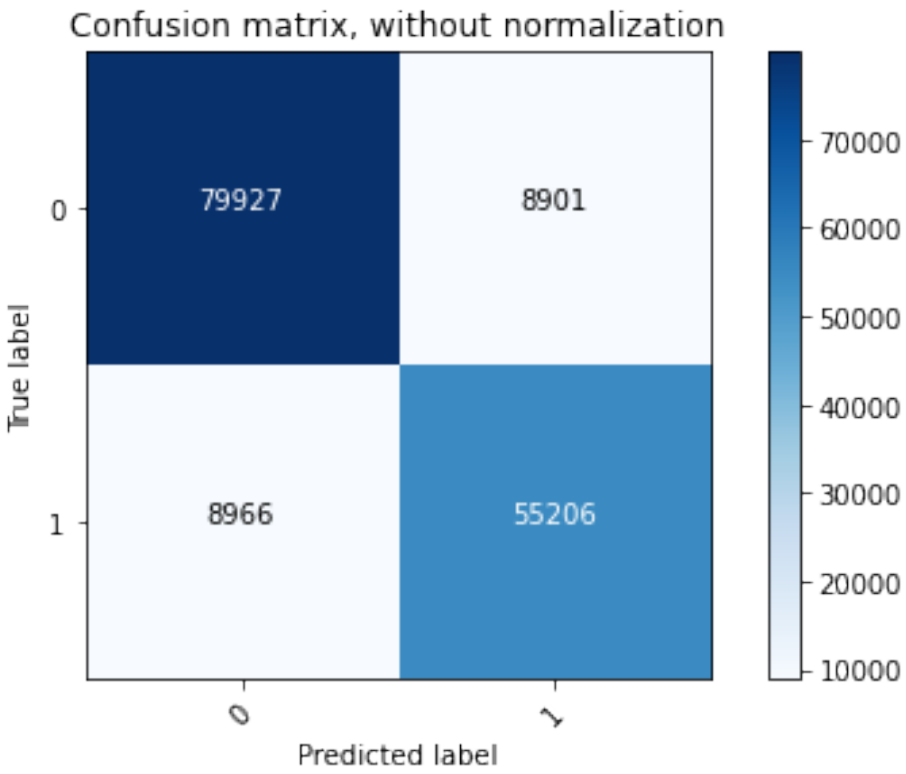


{"version_major":2,"version_minor":0,"model_id":"a4d6b4e1e43a4e20a95bd1ebfee459c3"}

Train loss: 0.08339353753071205, Train acc: 0.8175163398692811
Val loss: 0.3248643890023232, Val acc: 0.8640740513801575
Epoch: 3

{"version_major":2,"version_minor":0,"model_id":"c31905f8f2c24a18ab02e575a6a5e9e8"}

Confusion matrix, without normalization

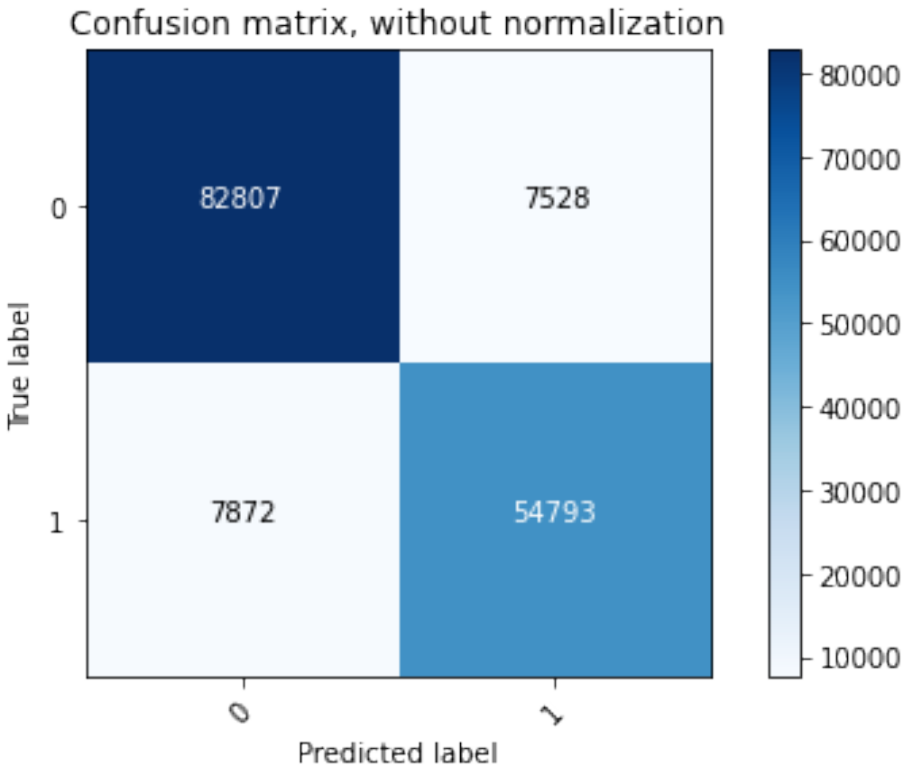


{"version_major":2,"version_minor":0,"model_id":"0ae42461685f4d76abaa4d7617fc9870"}

Train loss: 0.05506105837868709, Train acc: 0.8832222222222222
Val loss: 0.24542157277464866, Val acc: 0.899398148059845
Epoch: 4

{"version_major":2,"version_minor":0,"model_id":"0601970de67f43688c09f019a18f5ab3"}

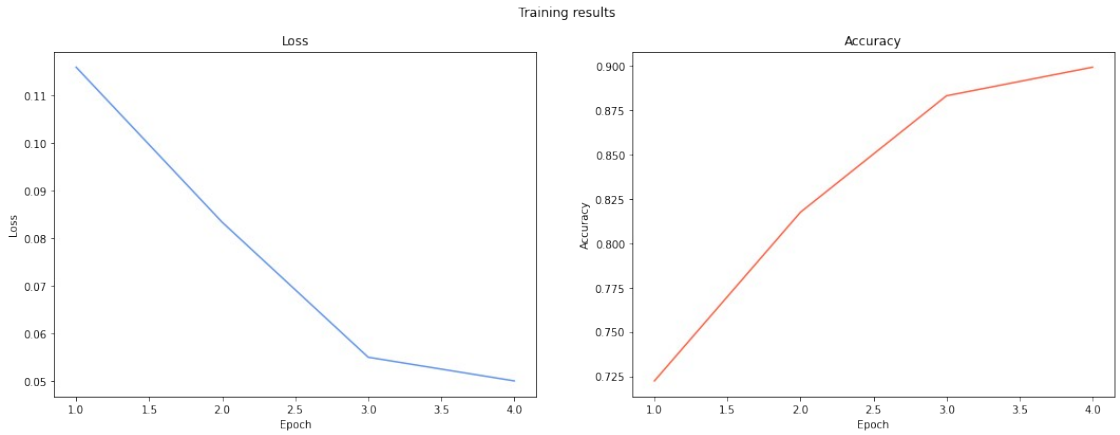
Confusion matrix, without normalization

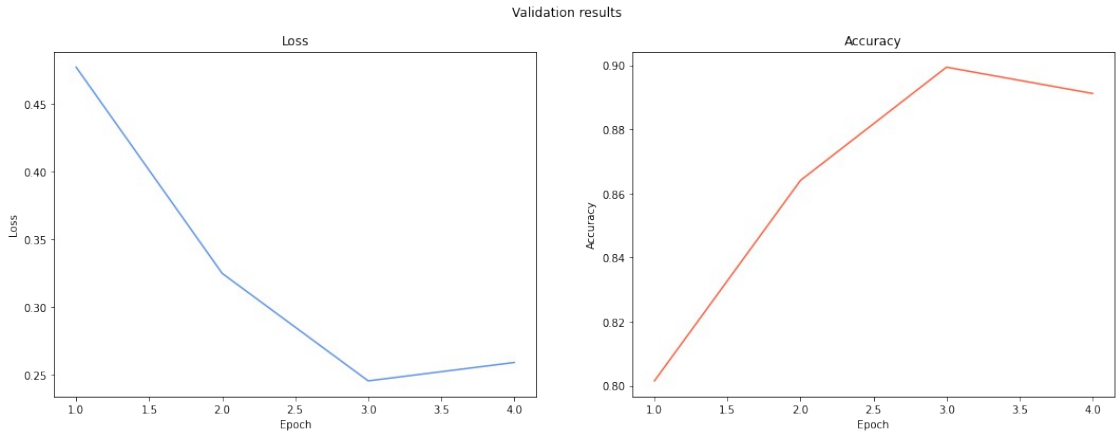


{"version_major":2,"version_minor":0,"model_id":"5532cc26b32e4730897912e8a9116266"}

```
Train loss: 0.05010055560691684, Train acc: 0.8993464052287582
Val loss: 0.2590277023613453, Val acc: 0.8912037014961243
```

```
plot_results(losses1, accs1, range(1, max_iters), title="Training results")
plot_results(losses2, accs2, range(1, max_iters), title="Validation results")
```





PREGUNTAS:

- Explique brevemente la diferencia entre una capa LSTM y una BLSTM (bidirectional LSTM).

La diferencia es que en el primer caso estamos utilizando información sobre instantes anteriores, de izquierda a derecha, mientras que en la bidireccional también utilizamos valores futuros contemplando los datos de derecha a izquierda.

- Incluya el código donde define *Model_1B* en el informe de la práctica.

Se incluye en el código, instanciándolo con parámetro `bidirectional=True`.

- ¿Qué modelo obtiene un mejor resultado sobre los datos de validación? ¿Por qué puede ocurrir esto?

Parece que el modelo que obtiene mejor resultado en validación es el primero, observando que en la última época decae bastante, si bien ambos son parecidos. Esto puede deberse a que el primer modelo necesita más épocas para ajustar, mientras que en el segundo caso tarda más en entrenar pero no necesita tantas épocas como el primero.

3.2. Modelo "más profundo"

En este apartado, vamos a partir nuevamente del modelo *Model_1* y vamos a añadir una segunda capa LSTM tras la primera, con el mismo tamaño y configuración, definiendo un nuevo modelo *Model_2*.

Entrénelo y compare los resultados.

```
class Model_2(nn.Module):
    def __init__(
        self, feat_dim: int = 20, hidden_size: int = 256,
bidirectional: bool = False
    ):
        super(Model_2, self).__init__()

        self.lstm = nn.LSTM(
```

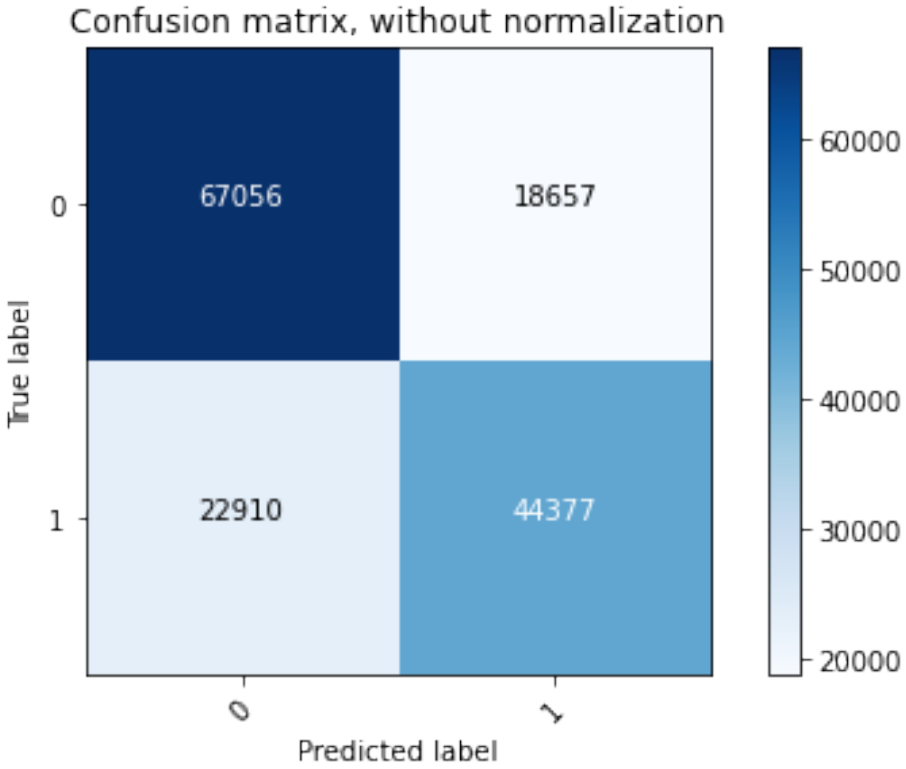
```
            input_size=feat_dim,
            hidden_size=hidden_size,
            batch_first=True,
            bidirectional=bidirectional,
        )
        self.lstm_2 = nn.LSTM(
            input_size=256,
            hidden_size=hidden_size,
            batch_first=True,
            bidirectional=bidirectional,
        )
        linear_size = hidden_size if not bidirectional else 2 *
hidden_size
        self.output = nn.Linear(linear_size, 1)

    def forward(self, x):

        out = self.lstm(x)[0]
        out = self.lstm_2(out)[0]
        out = self.output(out)
        out = torch.sigmoid(out)

        return out.squeeze(-1)

model = Model_2(feat_dim=20, bidirectional=False)
model = model.to(torch.device("cuda"))
print(model)

Model_2(
  (lstm): LSTM(20, 256, batch_first=True)
  (lstm_2): LSTM(256, 256, batch_first=True)
  (output): Linear(in_features=256, out_features=1, bias=True)
)

length_segments = 300

path_in_feat_train = "data/training_VAD/"
path_in_feat_val = "data/valid_VAD/"

criterion = nn.BCELoss()
optimizer = optim.Adam(model.parameters(), lr=0.001)

batch_size = 51
segment_sets_train = np.array_split(train_list, len(train_list) /
batch_size)
segment_sets_val = np.array_split(val_list, len(train_list) /
batch_size)

max_iters = 5
```

```
accs1, losses1 = [], []
accs2, losses2 = [], []

for epoch in range(1, max_iters):
    print("Epoch: ", epoch)

    t_loss, t_acc = train_phase(segment_sets_train, model,
path_in_feat_train)
    v_loss, v_acc = eval_phase(segment_sets_val, model,
path_in_feat_val)

    print("Train loss: {0}, Train acc: {1}".format(t_loss, t_acc))
    print("Val loss: {0}, Val acc: {1}".format(v_loss, v_acc))

    accs1.append(t_acc)
    accs2.append(v_acc.cpu())
    losses1.append(t_loss)
    losses2.append(v_loss)
```

```
Epoch:  1
```


```
{"version_major":2,"version_minor":0,"model_id":"d3b622db7d5241d28ee45
166a0943eb5"}
```


```
Confusion matrix, without normalization
```

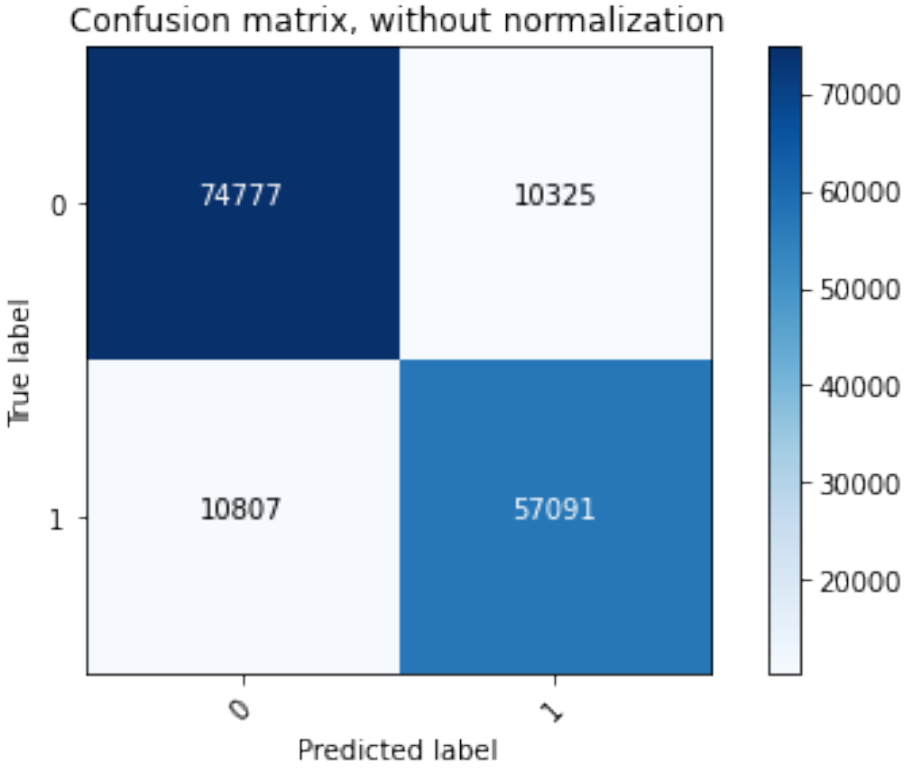


{"version_major":2,"version_minor":0,"model_id":"b51597853d45463f8c3de659b8fd42ee"}

```
Train loss: 0.1151085098584493, Train acc: 0.7283202614379085
Val loss: 0.4796378016471863, Val acc: 0.8029166460037231
Epoch:  2
```

{"version_major":2,"version_minor":0,"model_id":"7effde49f7a247028d65d35339129393"}

```
Confusion matrix, without normalization
```

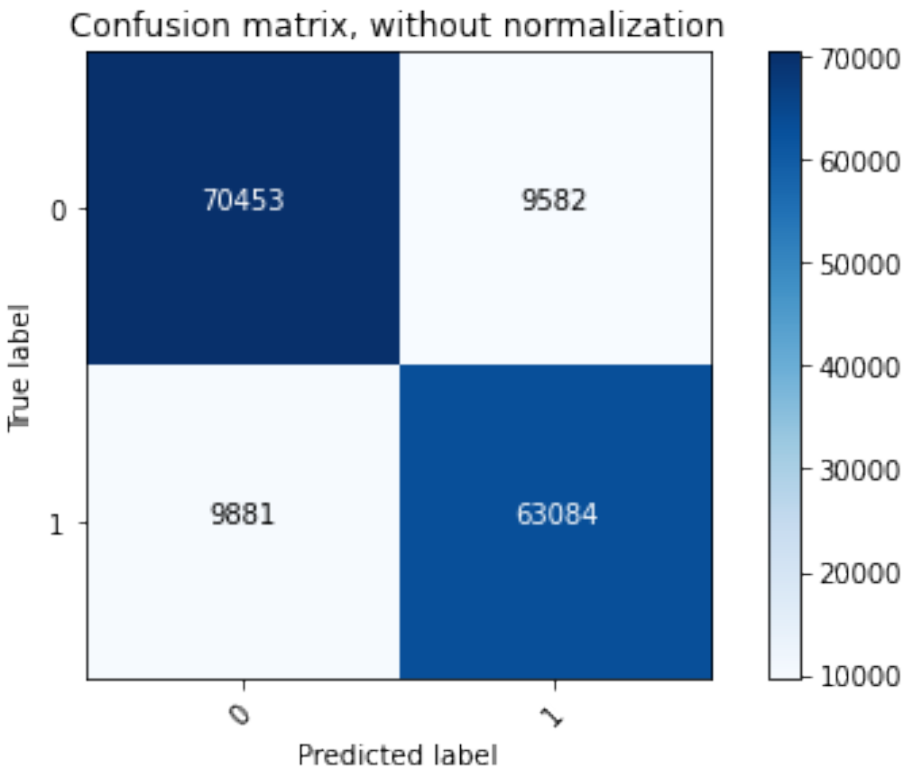


{"version_major":2,"version_minor":0,"model_id":"90e2f6979e854f8caa1622c5de6ecdff"}

```
Train loss: 0.07087305772538278, Train acc: 0.8618823529411764
Val loss: 0.35416449308395387, Val acc: 0.8443518280982971
Epoch:  3
```

{"version_major":2,"version_minor":0,"model_id":"929ea96a53b14e8c91fd4d607d050082"}

```
Confusion matrix, without normalization
```

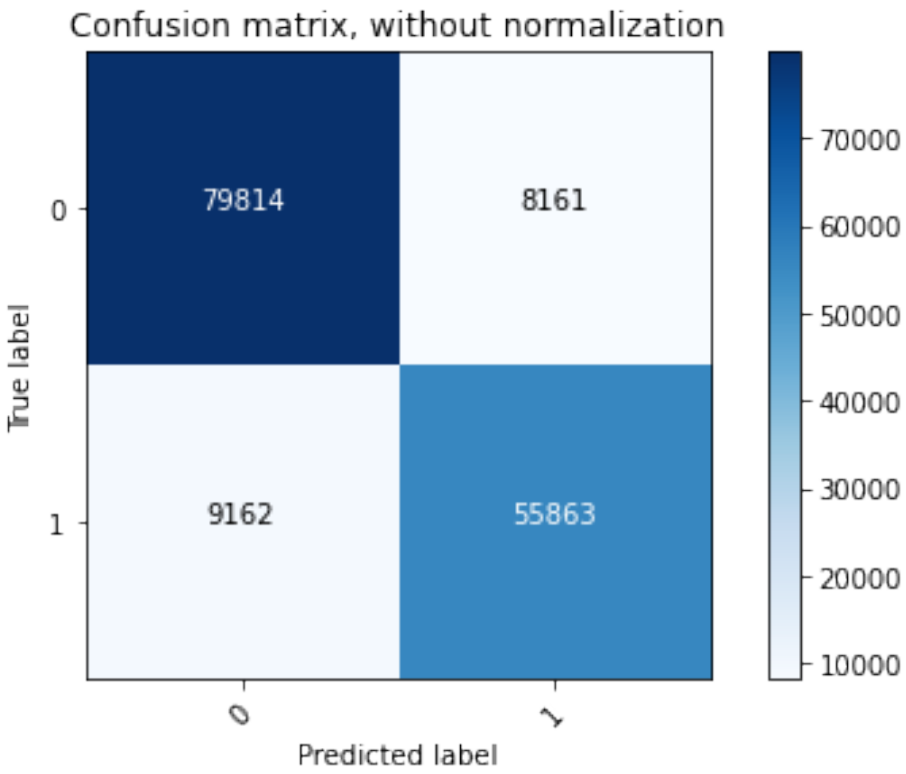


{"version_major":2,"version_minor":0,"model_id":"841f65fe00e84a38bda96b602de643fe"}

Train loss: 0.06456305349574369, Train acc: 0.8727908496732026
Val loss: 0.3348947986960411, Val acc: 0.8628240823745728
Epoch: 4

{"version_major":2,"version_minor":0,"model_id":"4b5dc83dfb694c308190257742481760"}

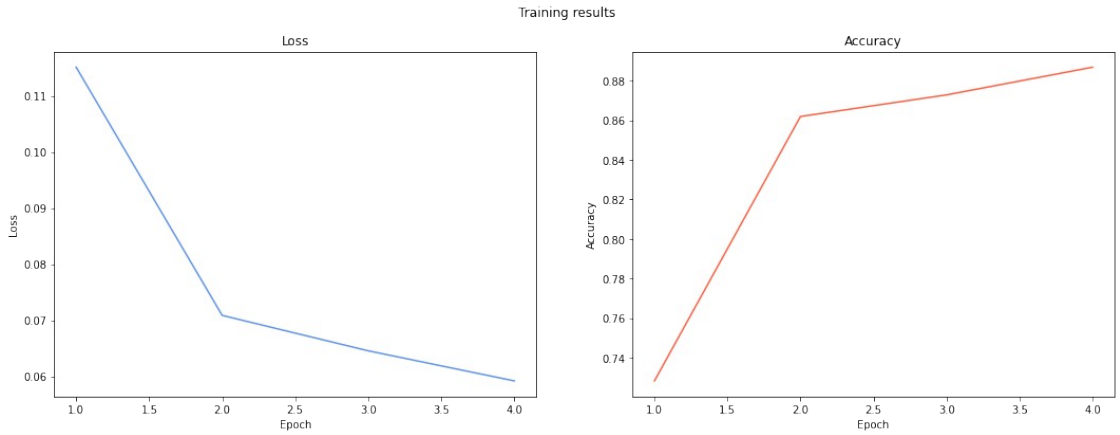
Confusion matrix, without normalization

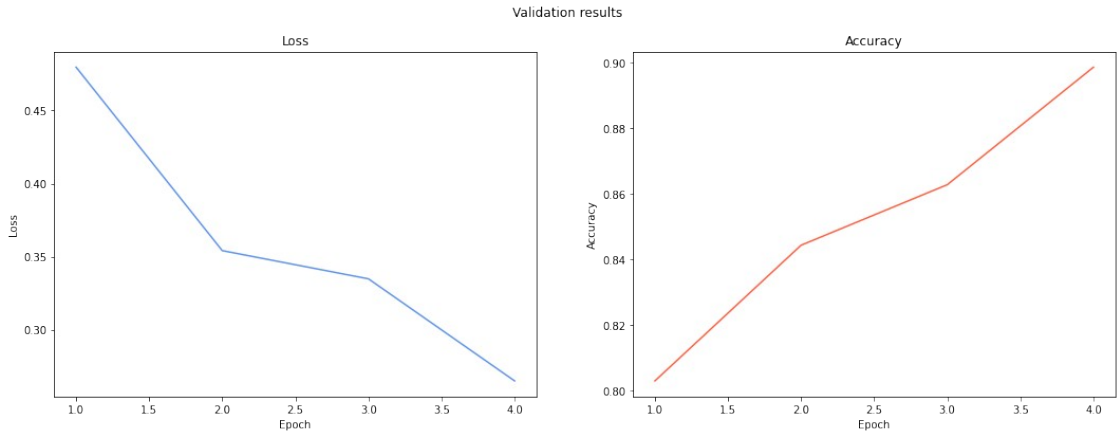


{"version_major":2,"version_minor":0,"model_id":"3f15a3fe9b3a46f6bc5409b65c512045"}

```
Train loss: 0.059196749154259175, Train acc: 0.8867777777777778
Val loss: 0.2650727108120918, Val acc: 0.8987036943435669
```

```
plot_results(losses1, accs1, range(1, max_iters), title="Training results")
plot_results(losses2, accs2, range(1, max_iters), title="Validation results")
```





PREGUNTAS:

- Incluya el código de la clase *Model_2* en la memoria.

Hecho.

- ¿Qué modelo obtiene un mejor resultado sobre los datos de validación, *Model_1* o *Model_2*? ¿Por qué puede ocurrir esto? Y con respecto a *Model_1B*, ¿cuál es mejor?

Aunque todos son bastante parecidos, parece que el primero es el mejor, ya que ofrece la mejor puntuación en validación. Se trata de una solución más sencilla, que no necesita tantos parámetros para entrenar como el último.

Si que es verdad que estamos valorando los resultados sin realizar ninguna búsqueda de mejores hiperparámetros y solo estamos valorando con un learning rate en concreto, un número de épocas fijo y con los mismos datos. Esto está bien para comparar bajo la misma "escala" pero estaría bien realizar más pruebas para determinar qué modelo es más a ciencia cierta mejor.